Password-based encryption in ZIP files

Dmitri Gabbasov

December 15, 2015

Abstract

In this report we give an overview of the encryption schemes used in the ZIP file format. We first give an overview of the file format itself and then describe three encryption schemes that can be used in ZIP files. Ultimately we will try to decide which scheme should be preferred.

1 Introduction

The ZIP file format was created in 1989 by PKWARE Inc. [2] and was first implemented in the company's PKZIP utility. The format has been documented in a specification document referred to as APPNOTE [1]. The earliest available version of APPNOTE (version 1.0) was written in 1990 [3]. The current version is 6.3.4, published in 2014.

The ZIP file format is widely known and is one of the most popular formats used to compress files. It is in fact a container format that supports a variety of compression algorithms, with *deflate* being the most common one.

2 ZIP file format

ZIP files are archives that store multiple files. Each file inside the archive is compressed individually, making it possible to extract them or add new ones without compressing or decompressing the entire archive. This contrasts with the format of compressed TAR archives, for which such random access is not possible.

Following is the structure of a ZIP archive containing n files.

Local file header 1
File data 1
Local file header 2
File data 2
⋮
Local file header n
File data n
Central directory file header 1
Central directory file header 2
⋮
Central directory file header n
End of central directory record

One should start reading a ZIP archive from the end. The *end of central directory record* looks as follows:

Offset	Size (bytes)	Description
0	4	Fixed value (`50 4b 05 06`)
4	2	Number of this disk
6	2	Disk where the central directory starts
8	2	Number of central directory records on this disk
10	2	Total number of central directory records
12	4	Size of the central directory (bytes)
16	4	Offset of the start of the central directory
20	2	Comment length (n)
22	n	Comment

There are certain fields that are related to the storing of ZIP archives on floppy disks, these are less relevant in modern days. The field at offset 16 allows one to navigate to the beginning of the central directory and proceed with reading the central directory file headers. Each central directory file header has the following structure.

Offset	Size (bytes)	Description
0	4	Fixed value (`50 4b 01 02`)
4	2	Version made by
6	2	Version needed to extract
8	2	General purpose bit flag
10	2	Compression method
12	2	File modification time
14	2	File modification date
16	4	CRC-32
20	4	Compressed size
24	4	Uncompressed size
28	2	File name length (n)
30	2	Extra field length (m)
32	2	File comment length (k)
34	2	Disk number where file starts
36	2	Internal file attributes
38	4	External file attributes
42	4	Offset of local file header
46	n	File name
$46 + n$	m	Extra field
$46 + n + m$	k	File comment

After reading all the central directory file headers, one knows exactly which files are present inside the archive. In each central directory file header, the field at offset 42 points to the location of the local file header inside the ZIP file. Navigating to that location allows one to proceed with reading the local file header, which has the following structure.

Offset	Size (bytes)	Description
0	4	Fixed value (`50 4b 03 04`)
4	2	Version needed to extract
6	2	General purpose bit flag
8	2	Compression method
10	2	File modification time
12	2	File modification date
14	4	CRC-32
18	4	Compressed size
22	4	Uncompressed size
26	2	File name length (n)
28	2	Extra field length (m)
30	n	File name
$30 + n$	m	Extra field

Some of the metadata that was present in the central directory file header is duplicated in the local file header for redundancy. The local file header is immediately followed by the (possibly) compressed contents of the file.

3 Encryption schemes used in ZIP files

The ZIP file format specification supports a number of encryption schemes. Originally, it only supported one – a custom scheme that has now been shown to be very weak. Since then, the specification has introduced the notion of *strong encryption*, which allows one to use ciphers such as RC4 and AES. However, there is an extension introduced by the company WinZip Computing (author of the WinZip archiver), that also uses AES to encrypt archives, but is not compatible with the *strong encryption* specification [4]. WinZip AES appears to be more popular than *strong encryption*.

All the encryption schemes that we cover in this report are password-based. As a result, they have several inherent drawbacks, like being susceptible to dictionary-attacks (albeit it is somewhat alleviated by slow key derivation algorithms).

3.1 Traditional PKWARE encryption

The original encryption scheme, commonly referred to as the PKZIP cipher, was designed by Roger Schaffely [1]. In [5] Biham and Kocher showed that the cipher is weak and demonstrated an attack requiring 13 bytes of plaintext. Further attacks have been developed, some of which require no user provided plaintext at all [6].

The PKZIP cipher is essentially a stream cipher, i.e. input is encrypted by generating a pseudo-random key stream and XOR-ing it with the plaintext. The internal state of the cipher consists of three 32-bit words: `key0`, `key1` and `key2`. These are initialized to `0x12345678`, `0x23456789` and `0x34567890`, respectively. A core step of the algorithm involves updating the three keys using a single byte of input, the update step looks as follows:

```
void update_keys(uint8 input) {
    key0 = crc32(key0, input);
    key1 = (key1 + (key0 & 0xff)) * 0x08088405 + 1;
    key2 = crc32(key2, key1 >> 24);
}

uint32 crc32(uint32 crc, uint8 input) {
    return (crc >> 8) ^ crctab[(crc & 0xff) ^ input];
}
```

where `crctab` is an array containing 256 fixed and known values. The code above should be interpreted as C code, the types `uint8`, `uint16` and `uint32` represent 8-bit, 16-bit and 32-bit unsigned integers, respectively.

Before encrypting (decrypting) a file, the cipher is first keyed by updating the three keys with the bytes of the user-supplied password as follows

```
for (uint8* b = password; b != 0; ++b) {
    update_keys(*b);
}
```

where `password` is an array of `uint8`s representing the bytes of the password and terminated by 0.

We now define the procedures needed to encrypt and decrypt a single byte:

```
uint8 stream_byte() {
    uint16 temp = key2 | 3;
    return (temp * (temp ^ 1)) >> 8;
}

uint8 encrypt_byte(uint8 plain_byte) {
    uint8 cipher_byte = stream_byte() ^ plain_byte;
    update_keys(plain_byte);
    return cipher_byte;
}

uint8 decrypt_byte(uint8 cipher_byte) {
    uint8 plain_byte = stream_byte() ^ cipher_byte;
    update_keys(plain_byte);
    return plain_byte;
}
```

Encryption is performed by successively calling `encrypt_byte` on each byte of the compressed file data. Similarly, decryption is performed by calling `decrypt_byte` successively on each byte of the encrypted file data.

If a file inside the ZIP archive is encrypted, the *general purpose bit flag* in the local file header and the central directory file header of that file will have its least significant bit set to 1.

Before encrypting a file in the archive, 12 random bytes are first prepended to its compressed contents and the resulting bytestream is then encrypted. Upon decryption, the first 12 bytes need to be discarded. According to the specification, this is done in order to render a plaintext

attack on the data ineffective.

The specification also states that out of the 12 prepended bytes, only the first 11 are actually random, the last byte is equal to the high order byte of the CRC-32 of the uncompressed contents of the file. This gives the ability to quickly verify whether a given password is correct by comparing the last byte of the decrypted 12 byte header to the high order byte of the actual CRC-32 value that is included in the local file header. This can be done before decrypting the rest of the file.

3.2 Strong encryption

Starting with version 5.0 (released in 2002) the PKZIP software was able to use other more common ciphers for encryption. However, the specification on how exactly these were implemented was not published immediately, but instead, in 2003. Even then, it remained vague, as we will see below.

In case password-based strong encryption is used, bits 0 and 6 in the *general purpose bit flag* are both set. Additionally, the *extra field* part of the file header is filled with the following structure:

Offset	Size (bytes)	Description
0	2	Fixed value (`17 00`)
2	2	Total size of all following fields (8)
4	2	Format of this structure, the only allowed value is 2.
6	2	Encryption algorithm identifier
8	2	Bit length of encryption key
10	2	Processing flags

The field at offset 6 can have the following values:

Value	Encryption algorithm
`01 66`	DES
`03 66`	3DES (168-bit key)
`09 66`	3DES (112-bit key)
`0E 66`	AES (128-bit key)
`0F 66`	AES (192-bit key)
`10 66`	AES (256-bit key)
`02 67`	RC2
`20 67`	Blowfish
`21 67`	Twofish
`01 68`	RC4

It is not clear how the field at offset 8 should be treated in case the key length is implied by the algorithm identifier, the obvious thing to do is to just ignore it.

The actual file data begins with yet another additional structure:

Offset	Size (bytes)	Description
0	2	Size of the initialization vector (n)
2	n	IV for this file
$n+2$	4	Total size of the following fields
$n+6$	2	Format of this structure, the only allowed value is 3
$n+8$	2	Encryption algorithm identifier
$n+10$	2	Bit length of encryption key
$n+12$	2	Processing flags
$n+14$	2	Size of encrypted random data (m)
$n+16$	m	Encrypted random data
$n+m+16$	4	Reserved, should be 0
$n+m+20$	2	Size of password validation data (k)
$n+m+22$	$k-4$	Password validation data
$n+m+k+18$	4	CRC-32 of password validation data

As can be seen, this structure duplicates all the information present in the *extra field*.

The specification does not go into the details of the encryption and decryption procedures. It only states that the block oriented ciphers operate in cipher block chaining (CBC) mode and provides the following pseudo-code for the encryption procedure:

```
Password = GetUserPassword()
MasterSessionKey = DeriveKey(SHA1(Password))
RD = CryptographicStrengthRandomData()
For Each File
    IV = CryptographicStrengthRandomData()
    VData = CryptographicStrengthRandomData()
    VCRC32 = CRC32(VData)
    FileSessionKey = DeriveKey(SHA1(IV + RD)
    ErdData = Encrypt(RD, MasterSessionKey, IV)
    Output = Encrypt(VData + VCRC32 + FileData, FileSessionKey, IV)
Done
```

where `VData` is *verification data* and `ErdData` is *encrypted random data* that are both included in the previously described structure. `Output` is the resulting ciphertext that is written into the ZIP file following the previous additional structure. Most notably however, the `DeriveKey` function is left undefined, which makes it hard to implement an encryption procedure that would be compatible with PKZIP. It s also not clear how the initialization vector is combined with stream ciphers like RC4.

3.3 WinZip AES encryption

The fact that the specification of *strong encryption* was initially not available and that PKZIP was the only software that could make use of it, caused WinZip to introduce its own way of creating ZIP archives that were encrypted using the AES cipher. A detailed specification for WinZip AES was published in 2003.

If WinZip AES is used to encrypt a file, the least significant bit of the *general purpose bit flag* is set to 1 and the *compression method* is set to 99 (decimal). The CRC-32 field is set to 0. The *extra field* is filled with the following structure:

Offset	Size (bytes)	Description
0	2	Fixed value (`01 99`)
2	2	Total size of the following fields (7)
4	2	Fixed value (`02 00`)
6	2	Fixed value (`41 45`)
8	1	AES key length identifier (1 - 128 bits, 2 - 192 bits, 3 - 256 bits)
9	2	The actual compression method used

A random salt is generated for each encrypted file. The salt is half the size of the AES key. The password and the salt are used to derive three keys with the Password-Based Key Derivation Function 2 (PBKDF2) with an iteration count of 1000. The first key is used to encrypt the compressed file contents using AES in counter (CTR) mode. The second one is used to compute a message authentication code (MAC) of the encrypted data using the HMAC-SHA1 algorithm. The third key is merely 2 bytes long and is used to facilitate a quick check of the correctness of the password. The WinZip AES specification links to third-party source code for exact implementation details.

The resulting file data, that is written into the ZIP file, is a concatenation of four things: the salt, the two byte key, the encrypted data and the MAC.

When decrypting, one first reads the salt, and uses it together with the password to generate the three keys. The third (two byte) key can now be used to do a quick check whether the password is correct, before decrypting the rest of the file. Finally, after decrypting the file, the MAC can be used to make sure that authenticity has not been compromised.

4 Summary

Out of the three reviewed encryption schemes, the traditional PKZIP cipher is by far the most used one. Some tools (InfoZIP, WinRAR, GNOME Archive Manager) do not even have the option to choose a different encryption scheme. Other tools (7-Zip) use the broken scheme by default, but can use a different one if the user requests so explicitly. 7-Zip only supports WinZip AES.

Sadly, the PKZIP cipher is very weak by today's standards. The official strong encryption remains a somewhat proprietary solution, but has a very good alternative in the form of WinZip AES. Thus, if one wants to maintain confidential and sensitive data inside a ZIP archive, one should refrain from using the default cipher. Instead, one should use either strong encryption or WinZip AES, with the latter one likely having higher availability.

References

[1] PKWARE Inc. *APPNOTE.TXT – .ZIP File Format Specification*, version 6.3.4
`https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT`

[2] PKWARE Inc. *.ZIP Application Note*
`https://www.pkware.com/support/zip-app-note/`

[3] PKWARE Inc. *APPNOTE.TXT*, version 1.0
`https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-1.0.txt`

[4] WinZip. *AES Encryption Information: Encryption Specification AE-1 and AE-2* http://www.winzip.com/aes_info.htm

[5] Biham E. and Kocher P. *A Known Plaintext Attack on the PKZIP StreamCipher*. Fast Software Encryption 2, Proceedings of the Leuven Workshop, LNCS 1008, December 1994

[6] Stay M. *ZIP Attacks with Reduced Known Plaintext*. Fast Software Encryption, LNCS 2355, 125–134. Springer-Verlag, 2002.